CommandReference

Updated Aug 26, 2010 by antirez

Redis Command Reference

Every command name links to a specific wiki page describing the behavior of the command.

Categorized Command List

Connection handling

Command	Parameters	Description
QUIT	-	close the connection
AUTH	*password*	simple password authentication if enabled

Commands operating on all value types

Command	Parameters	Description
EXISTS	*key*	test if a key exists
DEL	*key*	delete a key
TYPE	*key*	return the type of the value stored at key
KEYS	*pattern*	return all the keys matching a given pattern
RANDOMKEY	-	return a random key from the key space
RENAME	*oldname newname*	rename the old key in the new one, destroying the newname key if it already exists
RENAMENX	*oldname newname*	rename the *oldname* key to *newname*, if the *newname* key does not already exist
DBSIZE	-	return the number of keys in the current db
EXPIRE	-	set a time to live in seconds on a key
PERSIST	-	remove the expire from a key
TTL	-	get the time to live in seconds of a key
SELECT	*index*	Select the DB with the specified index
MOVE	*key dbindex*	Move the key from the currently selected DB to the *dbindex* DB
FLUSHDB	-	Remove all the keys from the currently selected DB
FLUSHALL	-	Remove all the keys from all the databases

Commands operating on string values

Command	Parameters	Description
SET	*key value*	Set a *key* to a string *value*
GET	*key*	Return the string value of the *key*
GETSET	*key value*	Set a key to a string returning the old value of the key
MGET	*key1 key2 ... keyN*	Multi-get, return the strings values of the keys
SETNX	*key value*	Set a key to a string value if the key does not exist
SETEX	*key time value*	Set+Expire combo command
MSET	*key1 value1 key2 value2 ... keyN valueN*	Set multiple keys to multiple values in a single atomic operation
MSETNX	*key1 value1 key2 value2 ... keyN valueN*	Set multiple keys to multiple values in a single atomic operation if none of the keys already exist
INCR	*key*	Increment the integer value of key
INCRBY	*key integer*	Increment the integer value of *key* by *integer*
DECR	*key*	Decrement the integer value of key
DECRBY	*key integer*	Decrement the integer value of *key* by *integer*
APPEND	*key value*	Append the specified string to the string stored at key
SUBSTR	*key start end*	Return a substring of a larger string

Commands operating on lists

Command	Parameters	Description
RPUSH	*key value*	Append an element to the tail of the List value at key
LPUSH	*key value*	Append an element to the head of the List value at key
LLEN	*key*	Return the length of the List value at key
LRANGE	*key start end*	Return a range of elements from the List at key
LTRIM	*key start end*	Trim the list at key to the specified range of elements
LINDEX	*key index*	Return the element at index position from the List at key
LSET	*key index value*	Set a new value as the element at index position of the List at key
LREM	*key count value*	Remove the first-N, last-N, or all the elements matching value from the List at key
LPOP	*key*	Return and remove (atomically) the first element of the List at key
RPOP	*key*	Return and remove (atomically) the last element of the List at key
BLPOP	*key1 key2 ... keyN timeout*	Blocking LPOP
BRPOP	*key1 key2 ... keyN timeout*	Blocking RPOP
RPOPLPUSH	*srckey dstkey*	Return and remove (atomically) the last element of the source List stored at *srckey* and push the same element to the destination List stored at *dstkey*

Commands operating on sets

Command	Parameters	Description
SADD	*key member*	Add the specified member to the Set value at key
SREM	*key member*	Remove the specified member from the Set value at key
SPOP	*key*	Remove and return (pop) a random element from the Set value at key
SMOVE	*srckey dstkey member*	Move the specified member from one Set to another atomically
SCARD	*key*	Return the number of elements (the cardinality) of the Set at key
SISMEMBER	*key member*	Test if the specified value is a member of the Set at key
SINTER	*key1 key2 ... keyN*	Return the intersection between the Sets stored at key1, key2, ..., keyN
SINTERSTORE	*dstkey key1 key2 ... keyN*	Compute the intersection between the Sets stored at key1, key2, ..., keyN, and store the resulting Set at dstkey
SUNION	*key1 key2 ... keyN*	Return the union between the Sets stored at key1, key2, ..., keyN
SUNIONSTORE	*dstkey key1 key2 ... keyN*	Compute the union between the Sets stored at key1, key2, ..., keyN, and store the resulting Set at dstkey
SDIFF	*key1 key2 ... keyN*	Return the difference between the Set stored at key1 and all the Sets key2, ..., keyN
SDIFFSTORE	*dstkey key1 key2 ... keyN*	Compute the difference between the Set key1 and all the Sets key2, ..., keyN, and store the resulting Set at dstkey
SMEMBERS	*key*	Return all the members of the Set value at key
SRANDMEMBER	*key*	Return a random member of the Set value at key

Commands operating on sorted zsets (sorted sets)

Command	Parameters	Description
ZADD	*key score member*	Add the specified member to the Sorted Set value at key or update the score if it already exist
ZREM	*key member*	Remove the specified member from the Sorted Set value at key
ZINCRBY	*key increment member*	If the member already exists increment its score by *increment*, otherwise add the member setting *increment* as score
ZRANK	*key member*	Return the rank (or index) or *member* in the sorted set at *key*, with scores being ordered from low to high
ZREVRANK	*key member*	Return the rank (or index) or *member* in the sorted set at *key*, with scores being ordered from high to low
ZRANGE	*key start end*	Return a range of elements from the sorted set at key
ZREVRANGE	*key start end*	Return a range of elements from the sorted set at key, exactly like ZRANGE, but the sorted set is ordered in traversed in reverse order, from the greatest to the smallest score

ZRANGEBYSCORE	*key min max*	Return all the elements with score >= min and score <= max (a range query) from the sorted set
ZCOUNT	*key min max*	Return the number of elements with score >= min and score <= max in the sorted set
ZCARD	*key*	Return the cardinality (number of elements) of the sorted set at key
ZSCORE	*key element*	Return the score associated with the specified element of the sorted set at key
ZREMRANGEBYRANK	*key min max*	Remove all the elements with rank >= min and rank <= max from the sorted set
ZREMRANGEBYSCORE	*key min max*	Remove all the elements with score >= min and score <= max from the sorted set
ZUNIONSTORE / ZINTERSTORE	*dstkey N key1 ... keyN* WEIGHTS *w1 ... wN* AGGREGATE SUM|MIN|MAX	Perform a union or intersection over a number of sorted sets with optional weight and aggregate

Commands operating on hashes

Command	Parameters	Description
HSET	*key field value*	Set the hash field to the specified value. Creates the hash if needed.
HGET	*key field*	Retrieve the value of the specified hash field.
HMGET	*key field1 ... fieldN*	Get the hash values associated to the specified fields.
HMSET	*key field1 value1 ... fieldN valueN*	Set the hash fields to their respective values.
HINCRBY	*key field integer*	Increment the integer value of the hash at *key* on *field* with *integer*.
HEXISTS	*key field*	Test for existence of a specified field in a hash
HDEL	*key field*	Remove the specified field from a hash
HLEN	*key*	Return the number of items in a hash.
HKEYS	*key*	Return all the fields in a hash.
HVALS	*key*	Return all the values in a hash.
HGETALL	*key*	Return all the fields and associated values in a hash.

Sorting

Command	Parameters	Description
SORT	*key* BY *pattern* LIMIT *start end* GET *pattern* ASC|DESC ALPHA	Sort a Set or a List accordingly to the specified parameters

Transactions

Command	Parameters	Description
MULTI/EXEC/DISCARD/WATCH/UNWATCH	-	Redis atomic transactions

Publish/Subscribe

Command	Parameters	Description
SUBSCRIBE/UNSUBSCRIBE/PUBLISH	-	Redis Public/Subscribe messaging paradigm implementation

Persistence control commands

Command	Parameters	Description
SAVE	-	Synchronously save the DB on disk
BGSAVE	-	Asynchronously save the DB on disk
LASTSAVE	-	Return the UNIX time stamp of the last successfully saving of the dataset on disk
SHUTDOWN	-	Synchronously save the DB on disk, then shutdown the server
BGREWRITEAOF	-	Rewrite the append only file in background when it gets too big

Remote server control commands

Command	Parameters	Description
INFO	-	Provide information and statistics about the server
MONITOR	-	Dump all the received requests in real time
SLAVEOF	-	Change the replication settings
CONFIG	-	Configure a Redis server at runtime

Comment by occmailbox, Apr 29, 2009

The Lists and Sets commands are fantastic!

Got one question: is there a lock command like "ACQUIRELOCK db1", "RELEASELOCK db1"?

This is very important for multiple data manipulations and every programming language and database supports it,

java got "synchronized" syntax, python got "thread.allocate_lock().acquire()", even php which does not support thread got "sem_acquire()".

In my project, we are using 2 languages, java and php, we used the mysql's "LOCK TABLES" instead the languages' locks.

If Redis got a support for this, or you have a good alternative way to do this. I would try Redis in my next project.

Thanks for the good work!

Comment by occmailbox, Apr 29, 2009

The "EXISTS" command cannot do what LOCK can do. Sometimes 2 different threads may call the "EXISTS" command at the same time, both returned the result "not exists" then both call the command "SET" to store the value, that's what exactly the php example "Retwis" register.php did (register.php used "GET" command but it's the same). I know that's only a example, and that can be fixed using php's own "sem_acquire()", I know, but now you are doing the cloud things, you cannot use a single language's lock on a single machine. Right? I suggest Redis bring 2 different level locks: the database lock and the key lock.

Comment by occmailbox, Apr 30, 2009

Now I figured out a way, use "SETNX" to acquire a lock, then after the manipulation, clear the lock. Is it the best way?

Comment by project member antirez, May 01, 2009

@occmailbox: Hello! The idea is that you can mount locking free algorithms on top of the atomic operation Redis provides. This is most of the times the best way to do things. Otherwise you can implement locking using SETNX or RENAMENX or other atomic primitives. Please post your specific problem on the Redis google group for more information. Almost all the times there is a locking free way to do stuff. Btw after Redis 1.0-stable release I plan to add locking primitives (that are not about keys or the whole DB but just about "tokens", so what a given token is really locking is up to your application). Ciao, Salvatore.

Comment by tobu...@gmail.com, May 01, 2009

The best way to do lock-free updates would be to have an equivalent to memcache CAS, a check and set operation. The client gets the old value and a cookie (the cookie can be the old value itself, or some last-modified timestamp or a version number). Then they compute the new value, and ask the server to perform a check and set. Either the server sets the value, or it has changed and the server returns a status code indicating no change was done. In which case, the client retries, starting at the GET stage.

Synopsis: CAS KEY NEW_VALUE COOKIE

Sets key to new value if it wasn't modified since COOKIE.

Comment by wong.edwin, Jul 24, 2009

I think there is a typo in the section **SUNION**. I believe the sentence ending "then this command produces the same result as SELEMENTS" should actually read "then this command produces the same result as **SMEMBERS**".

Comment by shein.artemiy, Jul 29, 2009

I don't understand where is LDEL command or something similar?

Comment by zze...@163.com, Sep 16, 2009

I think it should be better if add a EXPIRE command to DB, to let the whole db cache data expired

Comment by zze...@163.com, Sep 16, 2009

to support Map in the future？

Comment by birukoff, Sep 28, 2009

I don't see any practical benefit of having SCARD and LLEN as separate commands. Proposal: similar to SORT, which works for both sets and lists, unite them in one (for example, LEN).

Comment by acharnock, Oct 19, 2009

I echo Shein's question, where is LDEL command or similar?

Comment by project member antirez, Oct 20, 2009

@acharnock: it's not needed as DEL is already a vararg. DEL x y z ...

Comment by nova77, Oct 22, 2009

Is there a plan to support the memcache protocol? This one looks pretty close to me.

Comment by project member antirez, Oct 22, 2009

Hello nova77: no plans to support the memcached protocol. Redis has tons of features more, and a number of client libraries already implemented for many languages.

Comment by Bul...@gmail.com, Oct 24, 2009

Is the command list a reflection of what commands are available in the development version or stable released version ?

Comment by project member antirez, Nov 13, 2009

@Bulkan: in every man page of commands requiring Redis > 1.0 it's specified what version is needed.

Comment by alexgenaud, Dec 05, 2009

For completeness, shouldn't PING be added (NB: Shouldn't JRedis return a boolean?)

Comment by dr.marc.byrd, Feb 26, 2010

RFE: intersection of scored sets , which sums the scores of intersecting items. Also version of the command to write the result to another scored set.

Comment by peer.oded, Mar 11, 2010

Are there any plans to support "mincr" for multiple atomic increments?

Comment by jeethurao, Mar 18, 2010

@peer.oded: You're probably looking for INCRBY.

Comment by adam.skogman, Apr 12, 2010

Please, can you annotate the commands with the version needed? Hashes seem to need >1.2.x...

Comment by mtthwkfmn.gtalk, Apr 30, 2010

Redis is a good idea..... Fast db systems need central-like directory server things.

Comment by mtthwkfmn.gtalk, Apr 30, 2010

Redis is a good idea..... Fast db systems need central-like directory server things.

Comment by bortels, May 06, 2010

This is pretty awesome, gotta say. Having said that, here's a functionality gap that cropped up while working on my pew pew space game: there is no non-destructive copy or move, so far as I can see. I'd like to be able to copy the value of a key to a new key, across DBs for bonus points, without having to worry about the actual content. I'm doing it right now in code, but it's cumbersome for anything other than strings, and probably slower than need be (I've not tried it with big datasets yet). Or - maybe there's a clever way to do this that I missed? (Why? I'm using DBs as a poor-man's transaction - I set a busy flag for the client, copy the DB, do my work, copy it back, unset busy - and if something blows up halfway thru my work, I don't end up with an inconsistent DB).

Comment by yurasfromch, May 08, 2010

SORT is very powerful command when used with "GET pattern" but what if i don't need sorting? If some list, set or just string value holds part of key name of another value and is already sorted in necessary order (or produced

from previous SORT call)? Is any way to do the same as SORT? E.g. list A contains values 4,7,1,3 and I need to get key_4, key_7, key_1, key_3 in that order? Such feature is also very good for SUNION/SINTER/SDIFF to avoid storing result in some new key (even when result is stored, I cannot get what I need if I don't need sorting (because don't want additional overhead), but just need names of set members for building key names).

Comment by erwan.pigneul, May 10, 2010

Is there plans to address boolean data type. In fact, just storing a key without value, and accessing it with EXIST. I know it's already possible with value = 1 but my question could make sense if a optimized memory structure can be use to store it...

Thanks for your work, it's a way to open mind and find new way to address our requirements

Comment by yurasfromch, May 10, 2010

"ZRANK key member Return the rank (or index) or *member*" must be changed to "ZRANK key member Return the rank (or index) OF *member*" and similar for ZREVRANK

Comment by kristopolous, Jul 16, 2010

Their are a few typos. How does one edit this?

Comment by project member michaelrbernstein, Jul 16, 2010

I think I caught all the typos. Post any others you find please.

Comment by herstik, Jul 28, 2010

I think that the sorted sets are very powerful for managing priority queues, but it lacks a lot of useful commands which regular sets have, like SMOVE, SPOP etc.

Comment by shariefsk, Aug 03, 2010

Is there a way I can multi get keys with the key created at (or expires at timestamp if expiry is set) timestamp? What I need is TTL like method, but for multiple keys in one atomic operation.

Comment by dvsoftware, Aug 12, 2010

There is no parameter for AUTH in the list. It should be 'password'.

Comment by project member michaelrbernstein, Aug 22, 2010

Thanks, all set on the AUTH fix.

Comment by jorangreef, Sep 08, 2010

Is it possible to do a SUNION or SINTERSECTION on a set of keys, pass the result of this directly to the MGET command and then return the result of the MGET command?

This would save tremendous amounts of network traffic, when doing set operations on indexes with the intention to return the objects represented by the results of these set operations.

One could just pass in a reference to the indexes, and then have the objects returned in one step.

Enter a comment:

Submit

Wiki markup help hide

=Heading1=
==Heading2==
===Heading3===

bold _italic_
`inline code`
escape: `*`

Indent lists 2 spaces:
 * bullet item
 # numbered list

{{{
verbatim code block
}}}

Horizontal rule

WikiWordLink
[http://domain/page label]
http://domain/page

|| table || cells ||

More examples